CSE 559A: Computer Vision

[credit: danjodon.deviantart.com]

Fall 2017: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Staff: Abby Stylianou (abby@wustl.edu), Jarett Gross (jarett@wustl.edu)

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Nov 2, 2017

GENERAL

- PSET 4 posted Tuesday.

- Will require use of the census function from last PSET 3.

- Recitation next Friday November 10.

Upcoming Events:

- CSE Fall research day this Friday
- WiCS Event: Hidden Figures Awareness: Nov 15

Checkout CSE website.

GROUPING & SEGMENTATION

SLIC

$$L = \arg\min_{L} \min_{\{\mu_k\}} \sum_{k=1}^{K} \sum_{n:L[n]=k} \|I'[n] - \mu_k\|^2$$

- Begin with some initial assignment $L[n]$.
- At each iteration ...

Step 1: For each k, assign

$$\mu_k = \text{Mean}\{I'[n]\}_{L[n]=k}$$

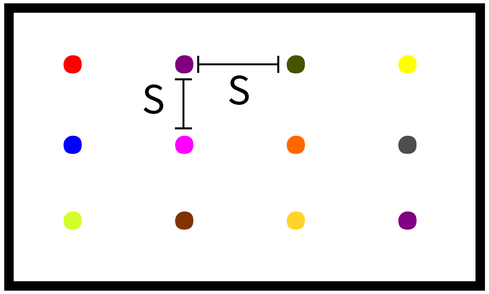
Step 2: For each n, assign

$$L[n] = \arg\min_{k} \|I'[n] - \mu_k\|^2$$

- How do we initialize ?
- Do we really need to do $K \times N$ computations of $\|I'[n] - \mu_k\|^2$?

GROUPING & SEGMENTATION

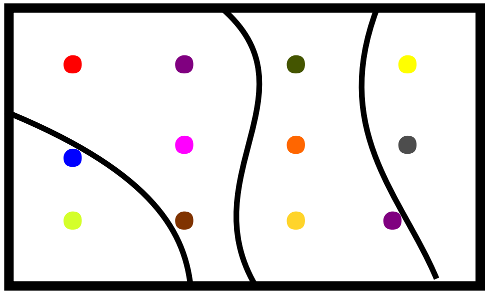
SLIC: Initialization



- Actually, begin with an assignment of $\{\mu_k\}$ (and do a step 2).
- Given desired number of super-pixels K, choose K points on a grid.
  - Spaced horizontally and vertically apart by $S = \sqrt{\frac{HW}{K}}$
- Set each $u_k = I'[n_k]$ as the augmented vector of one of these points.
- In step 2, each seed is going to attract pixels in its neighborhood that are most like it.

GROUPING & SEGMENTATION

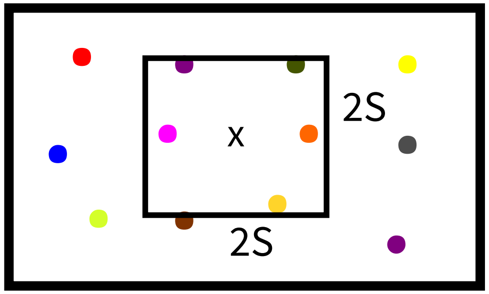
SLIC: Initialization



- Actually, begin with an assignment of $\{\mu_k\}$ (and do a step 2).
- Given desired number of super-pixels K, choose K points on a grid.
  - Spaced horizontally and vertically apart by $S = \sqrt{\frac{HW}{K}}$
- Set each $u_k = I'[n_k]$ as the augmented vector of one of these points.
- In step 2, each seed is going to attract pixels in its neighborhood that are most like it.
- Sometimes this initialization gives you a 'seed' that lies right on an edge.
  - Bad because pixel on either side of edge will often look nothing like it.
- Solution: Look in a 3x3 neighborhood, and choose pixel with lowest gradient magnitude.

GROUPING & SEGMENTATION

SLIC: Minimization

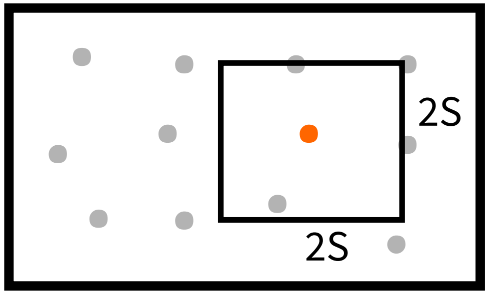


At any given iteration, for step 2:

- Don't consider all possible K for every n.
- Instead, say that a pixel n can only be assigned to a cluster k if n is within a $2S \times 2S$ window around the spatial co-ordinates in u_k.
- Note that μ_k's will no longer be on a regular grid.

GROUPING & SEGMENTATION

SLIC: Minimization



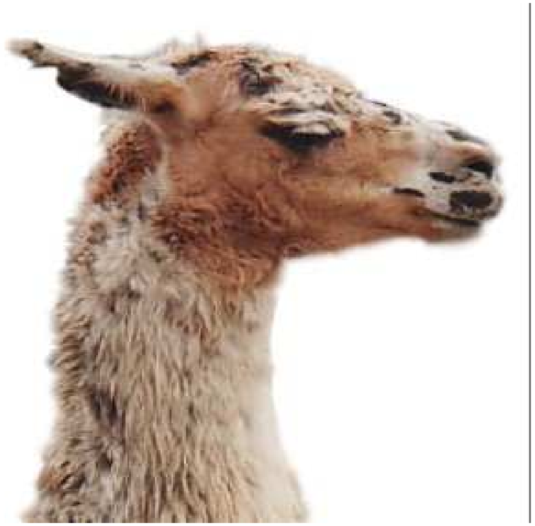
At any given iteration, for step 2:

- Initialize min_dist[n] to Infinity for all n
- Loop through each u_k, and consider pixels in $2S \times 2S$ window around μ_k
    - This will be a regular grid.
- For each pixel in this window, compute distance of $I'[n]$ to μ_k,
  compare to min_dist[n], if lower, update min_dist[n] and update L[n].

Do we need to loop over K ? Can get some parallelism if you're clever about it.

GROUPING & SEGMENTATION

Graph-based Methods

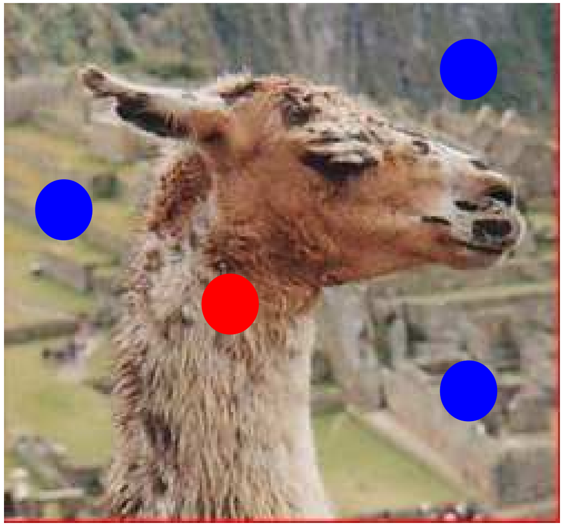


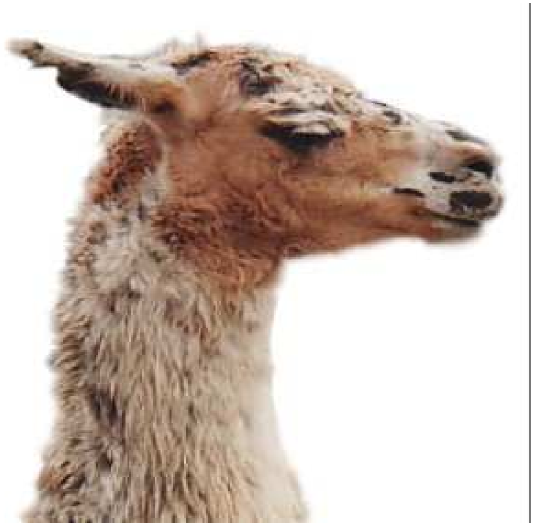
Foreground / Background Segmentation

Image from Rother et al., GrabCuts.

GROUPING & SEGMENTATION

Graph-based Methods

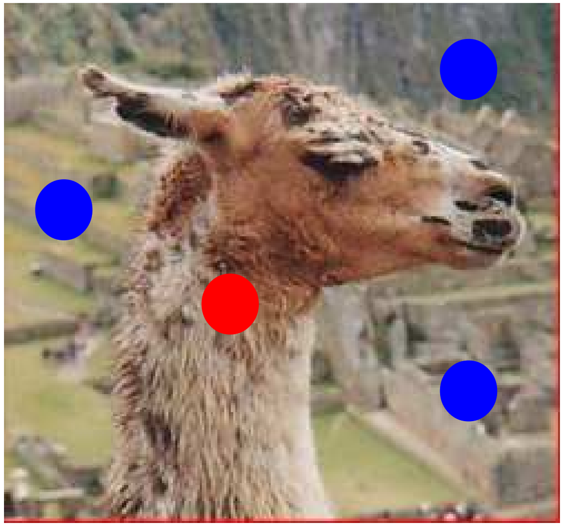


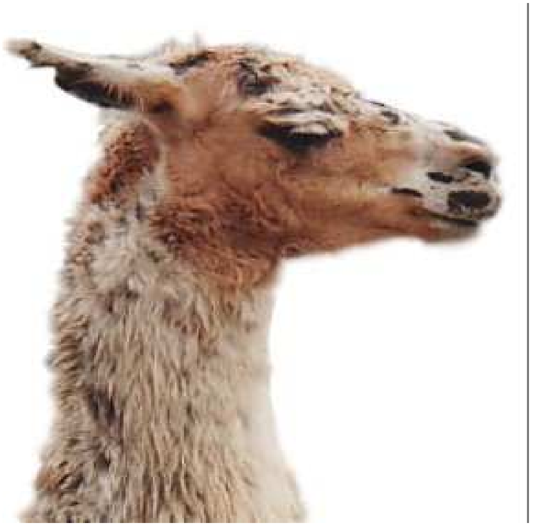


Assign a label of 1 (foreground) or 0 (background) for each pixel in the image.

Let's say user has labeled some pixels as foreground or background.
(or these are noisy / sparse predictions from some algorithm)

Graph-based Methods



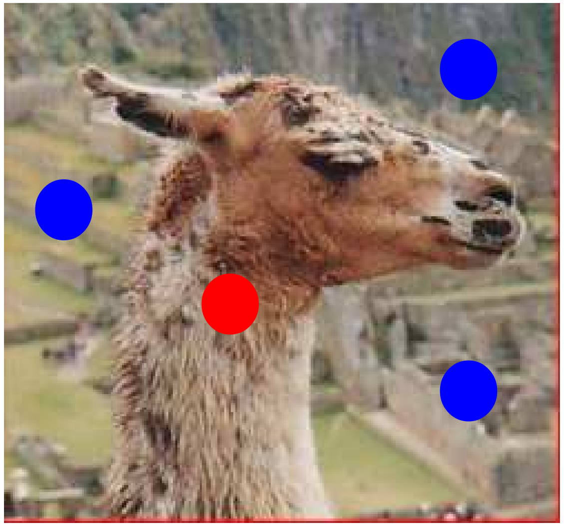


$$L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in \mathbb{E}} S_{n,n'}(L[n], L[n'])$$

Kind of like our stereo setup, but binary labeling problem.

GROUPING & SEGMENTATION

Graph-based Methods



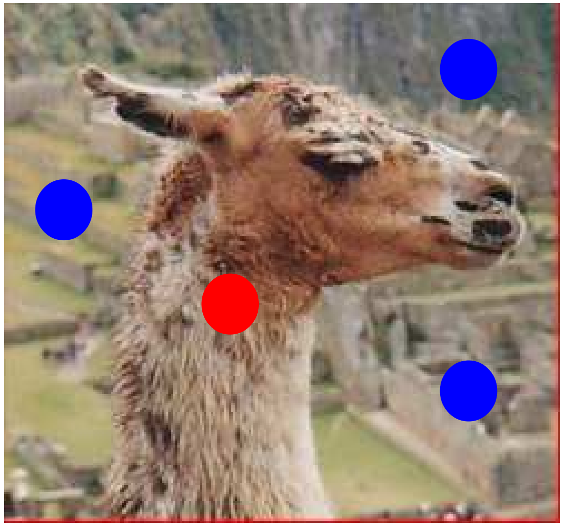
$$L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in \mathbb{E}} S_{n,n'}(L[n], L[n'])$$

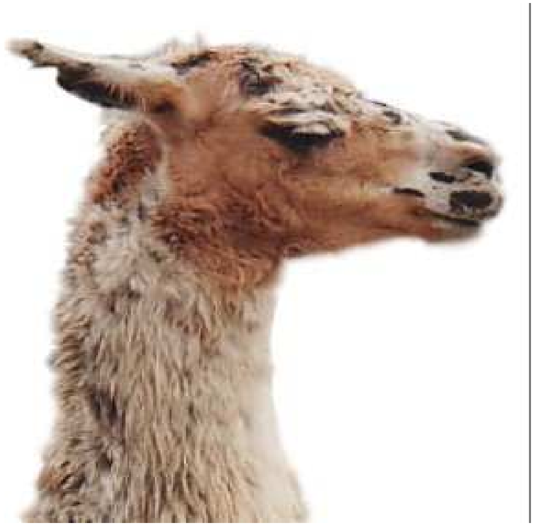
Comes from user input / algorithm

E.g., 0 for unlabeled pixels.
Very high / infinite cost at n for L[n]
different from user label

GROUPING & SEGMENTATION

Graph-based Methods



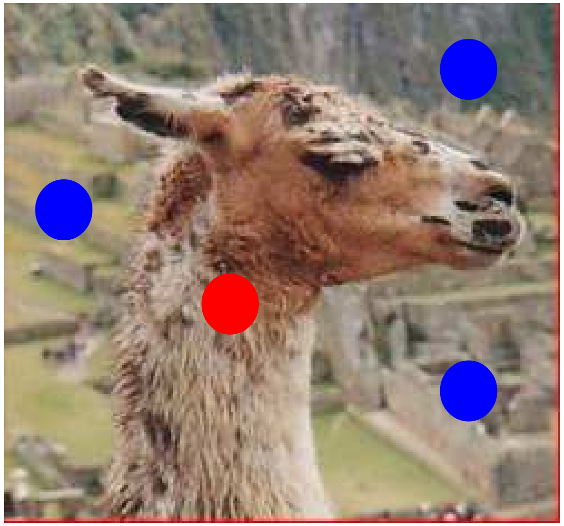


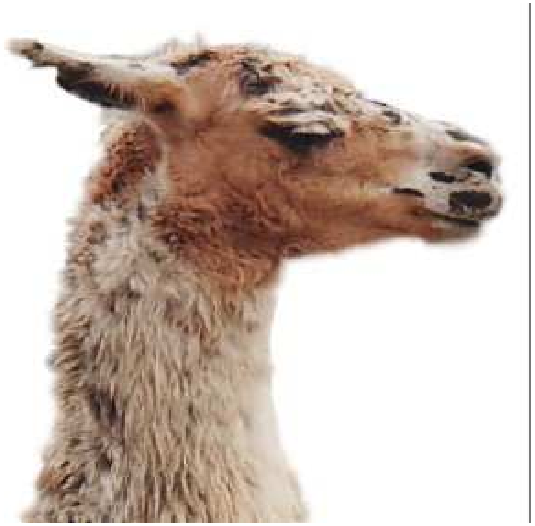
$$L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in \mathbb{E}} S_{n,n'}(L[n], L[n'])$$

Again, pairs of neighboring pixels.
Horizontal / Vertical / Diagonal

GROUPING & SEGMENTATION

Graph-based Methods



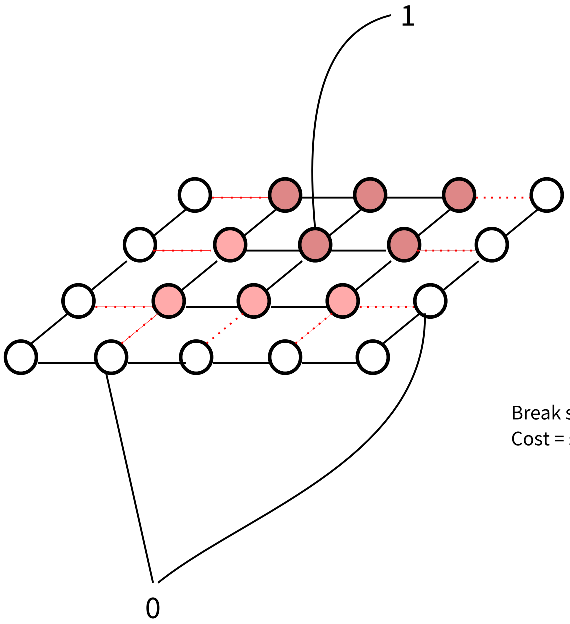


$$L = \arg \min_{L[n]\in\{0,1\}} C[n, L[n]] + \sum_{(n,n')\in\mathbb{E}} S_{n,n'}(L[n], L[n'])$$

Now will depend on pixel location.
Often based on intensity differences / whether there is an edge.

GROUPING & SEGMENTATION

Graph-based Methods



Break set of vertices into two disconnected graphs by removing edges.
Cost = sum of weight on edges you removed

$$L = \arg \min_{L[n] \in \{0,1\}} C[n, L[n]] + \sum_{(n,n') \in \mathbb{E}} S_{n,n'}(L[n], L[n'])$$

GROUPING & SEGMENTATION

- Min-cut Problem: Can be solved exactly and efficiently
- Different variant: Normalized Cuts, where you also care about the size of each segment.
    - Doesn't allow you to minimize cost by choosing a corner point and breaking small number of edges.
- Generalization to multi-label cases. (Has even been used for stereo!)

References

- Boykov and Kolmogorov, An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, PAMI 2004.
- Delong et al., Fast Approximate Energy Minimization with Label Costs, IJCV 2012.
- Rother et al., GrabCut -Interactive Foreground Extraction using Iterated Graph Cuts, SIGGRAPH 2004.

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

So far, given an input X and desired output Y we have

- Tried to explain the relationship of how X results from Y
  - X = observed image(s) / Y = clean image, sharp image, surface normal, depth
  - Noise, photometry, geometry, ...
- Often put a hand-crafted "regularization" cost to compute the inverse
  - Depth maps are smooth
  - Image gradients are small
- But sometimes, there is no way to write-down a relationship between X and Y ?
  - X = Image, Y = Does the image contain a dog ?
- Even if there is, the hand-crafted regularization cost is often arbitrary.
  - Real images contain far more complex and subtle regularity.

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of y conditioned on a specific value of x,
  - To minimize some notion of "risk" or "loss"

Define a loss function $L(y, \hat{y})$, which measures how much we dislike $\hat{y}$ as our estimate, when y is the right answer.

Examples

- $L(y, \hat{y}) = ||y - \hat{y}||^2$
- $L(y, \hat{y}) = ||y - \hat{y}||$
- $L(y, \hat{y}) = 0$ if $y = \hat{y}$, and some C otherwise.

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of y conditioned on a specific value of x,
  - To minimize some notion of "risk" or "loss"

Ideally,

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \; dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y') dy'}$$

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \ \ dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

- So we have a loss (depends on the application)
- We can compute $P(y|x)$ from P_{XY}.
- But we don't know P_{XY} !

Assume we are given as training examples, samples $(x, y) \sim P_{XY}$ from the true joint distribution.

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \ \ dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

Given $\{(x_i, y_i)\}$ as samples from P_{XY}, we could:

- Estimate P_{XY}
  - Choose parametric form for the joint distribution (Gaussian, Mixture of Gaussians, Bernoulli, $\ldots$)
  - Estimate the parameters of that parametric form to "best fit" the data.
  - Depending again on some notion of fit (often likelihood)

$$P_{XY}(x, y) = f(x, y, ; \theta)$$

$$\theta = \arg\max_{\theta} \sum_i \log f(x_i, y_i; \theta)$$

Maximum Likelihood Estimation

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \ \ dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

$$P_{XY}(x, y) = f(x, y, ; \theta)$$

$$\theta = \arg\max_{\theta} \sum_i \log f(x_i, y_i; \theta)$$

So that's one way of doing things ...

- You're doing a minimization for learning P_{XY}, but then also a minimization at "test time" for every input x.
- You're approximating P_{XY} with some choice of the parametric form f.
- And it's possible that the best θ that maximizes likelihood, may not be the best θ that minimizes loss.

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the loss $L(y, f(x))$ is low.

$$f = \arg\min_f \int \left(\int L(y, f(x)) p(y|x) dy \right) p(x) dx$$

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the loss $L(y, f(x))$ is low.

$$f = \arg\min_f \int\int L(y, f(x)) \; p_{XY}(x, y) dx dy$$

What we're going to is to replace the double integration with a summation over samples !

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the loss $L(y, f(x))$ is low.

$$f = \arg\min_f \sum_i L(y_i, f(x_i))$$

What we're going to is to replace the double integration with a summation over samples !

Empirical Risk Minimization

- So instead of first fitting the probability distribution from training data, and then given a new input, minimizing the loss under that distribution ...
- We are going to do a search over possible functions that "do well" on the training data, and assume that a function that minimizes "empirical risk" also minimizes "expected risk".

MACHINE LEARNING: CRASH COURSE / QUICK RECAP

Next time:

- How do you choose the space of possible functions to minimize over ?
- What are the consequences of this to the expected error ?
- How do you solve the optimization problem, efficiently ?